# Guía rápida sobre ejecución de especificaciones algebraicas en Maude bajo el entorno Eclipse para estudiantes de Estructuras de Datos.
## Actualizado para poder utilizar la herramienta de *testing*.

Isabel Pita
Dept. Sistemas Informáticos y Computación
Universidad Complutense de Madrid.
(Última modificación 15-01-2011)

## 1. Introducción

El presente manual es una guía para ayudar a los estudiantes de la asignatura de Estructuras de Datos y de la Información de segundo curso de Ingeniería Superior en Informática e Ingeniería Técnica en Informática de Sistemas de la UCM a ejecutar especificaciones algebraicas en lenguaje Maude. El lenguaje Maude está disponible para Windows, Linux y Mac. Bajo Linux y Mac lo más cómodo es ejecutar Maude desde la línea de comandos y utilizar un editor de texto para modificar los ficheros. Para ejecutar Maude en Windows, se han creado unos *plugins* que permiten ejecutar Maude bajo el entorno Eclipse.

El manual se compone de:

1. Como instalar Maude, bajo Linux, Mac y Windows
2. Como instalar Eclipse para ejecutar Maude en Windows.
3. Como definir una especificación algebraica sencilla y como ejecutarla.
4. Conceptos básicos de Maude.

Este manual ha sido actualizado para poder utilizar la herramienta de *testing* para probar implementaciones en C++ de los tipos de datos con las especificaciones realizadas. Los principales cambios respecto a la versión anterior son: el entorno Eclipse utilizado (sec. 3) y el uso del lenguaje *Full Maude* en lugar del *Core Maude*. La principal diferencia entre ambos lenguajes es que en *Full Maude* los módulos deben escribirse entre paréntesis y se añade un identificador a las ecuaciones. El manual para ejecutar la herramienta de *testing*: *Manual para realizar* testing *de TADs especificados en Maude e implementados en C++* se proporciona por el Campus Virtual.

## 2. Instalar Maude

Para ejecutar Maude desde Linux y Mac se puede descargar el sistema gratuitamente desde la página principal de Maude: `http://maude.cs.uiuc.edu`, en el apartado *download latest version of Maude 2*. Para especificar tipos abstractos de datos es suficiente con descargar el sistema *Core Maude 2.5*. Las facilidades proporcionadas por *Full Maude* se utilizan en la generación de casos de prueba, pero no es necesario bajarlas de este sitio web. En esta página se encuentra también disponible el manual completo del sistema, un tutorial y otra documentación sobre el lenguaje Maude.

Para instalar Maude bajo Windows se recomienda utilizar el instalador desarrollado en la Universidad de Valencia dentro del proyecto MOMENT. Se puede descargar gratuitamente desde la página `http://moment.dsic.upv.es`, dentro del apartado *Maude for Windows*. Descargad la versión Maude for Windows 2.5, utilizando el botón *download*. Al realizar la instalación modificar la ruta en la que se instalará el ejecutable para que **no se utilicen directorios cuyo nombre tenga espacios**, estos pueden dar problemas al ejecutar Maude desde Eclipse y el *testing*.

Si ya se tenía una versión instalada de Maude para Windows debe desinstalarse con el programa que se proporcionaba con la instalación, antes de proceder a la nueva instalación.

Copiar el fichero `dd.maude`, que se proporciona a través del Campus Virtual, en el directorio donde se encuentre el ejecutable de Maude. Este fichero es necesario para ejecutar las especificaciones con la sintaxis de *Full Maude* y poder generar los casos de prueba. La sintaxis de *Full Maude* es muy semejante a la de *Maude* y será la utilizada en todas las especificaciones que se realicen este curso. En caso de no tener acceso al Campus Virtual el alumno debe ponerse en contacto con la profesora de la asignatura para que le proporcione el fichero.

## 3. Instalar Eclipse IDE for C/C++ Developers con la opción de ejecutar Maude y los casos de prueba.

En cursos anteriores se ha utilizado Eclipse Classic 3.5.0 como entorno de trabajo para Maude. Este curso se utiliza el entorno Eclipse IDE for C/C++ para tener un entorno integrado de ejecución de especificaciones Maude e implementaciones en C++. Este nuevo entorno se puede utilizar tanto para ejecutar especificaciones Maude como para hacer el *testing*. Si se quiere seguir utilizando el entorno anterior, solo se podrán ejecutar especificaciones Maude. En este caso es preferible utilizar el manual del curso anterior.

A continuación nos centramos en la instalación de Eclipse para el entorno integrado.

Para poder ejecutar Maude desde Eclipse es necesario tener instalado el JDK de Java version 6.0. Si no se tiene instalada, se puede descargar gratuitamente desde la página `http://www.java.com`

El entorno Eclipse IDE for C/C++ Developers se puede descargar desde la página http://www.eclipse.org/downloads. Se debe seleccionar el sistema operativo que se vaya a utilizar.

Una vez descargado Eclipse, para poder ejecutar Maude deben descargarse las herramientas de desarrollo para Maude del proyecto MOMENT.

Para ello:

- ejecutar Eclipse desde el directorio en que se ha cargado;
- seleccionar un espacio de trabajo donde se guardarán los proyectos (conjunto de módulos de una especificación, implementaciones en C++ y ficheros de texto) que se realicen. **Si estas utilizando Windows no debes usar un path cuyos directorios tengan blancos**;
- seleccionar en el menú *Help*, la opción *Install new software*. En el apartado *work with* pulsar el botón *Add*. Seleccionar un nuevo sitio con el nombre que se desee (p.e. MOMENT Public Update Site) y dirección `http://moment.dsic.upv.es/updates/`. Seleccionar `MDT` e instalarlo.

Este proceso está explicado en detalle en el manual de instalación desarrollado por el proyecto MOMENT, aunque la explicación anterior debería ser suficiente para la mayor parte de los sistemas. Página `http://moment.dsic.upv.es`; seleccionar *Maude development tools*; seguir el hiperlink *InfoCenter*; seleccionar *Maude Development Tools*; seleccionar *Installation Process*.

A continuación se debe seleccionar el entorno de ejecución para Maude.

- En el menú *Window* de Eclipse seleccionar *Show View*, a continuación, en *Other* seleccionar *Maude* y a continuación *Maude Console*. La ventana de Eclipse se encuentra ahora dividida en varias partes. Si no lo está, cerrar la consola que se acaba de abrir y aparecerá. Las siguientes veces que se abra Eclipse bastará con teclear en la ventana de bienvenida y ya se cargarán las ventanas.

  Las ventanas que se van a utilizar son:

  - Una ventana denominada *Project Explorer*. En esta ventana se muestran los proyectos existentes en el espacio de trabajo seleccionado al abrir Eclipse, y los archivos de cada proyecto.
  - El editor, donde se escribe el contenido de los módulos.

    - La consola de Maude, con los botones para ejecutar y detener el programa. Si se ha minimizado volver a abrirla desde *Show View*, *Other*, *Maude* y colocarla debajo de la ventana del Editor.Tambien se podrá abrir la consola cuando se cree un fichero `.maude` utilizando un botón de la barra de herramientas.
    - La consola de C++.

  El resto de las ventanas pueden minimizarse.

- Para poder ejecutar Maude hay que indicar al sistema donde se encuentra el ejecutable. Para ello seleccionar en el menú *Window* de Eclipse la opción *Preferences* y aquí la línea *Maude Preferences*.
    - Buscar el path en el que se encuentra el ejecutable de Maude que se ha cargado en el punto 2.1 y añadirlo en el apartado *Maude binary file* (se puede utilizar el browser que hay en la ventana).
    - Indicar el path del fichero `dd.maude` en el apartado de *Full Maude File*. El fichero debe encontrarse en el mismo directorio que el ejecutable de Maude.
    - Hay que indicar un directorio donde guardar los ficheros temporales (log). Como en el caso del workspace no utilizar directorios cuyo nombre tenga espacios. Si se están utilizando los ordenadores de la Facultad utilizad un directorio con permiso de escritura.
    - Marcar la opción de *Run Full Maude*.
    - En la opcion *Advance* marcar *Other* en *Operating System* y escribir el siguiente comando `cmd.exe , /C , bash -c ''echo $$; exec %maudebin% %fullmaude% -interactive -no-tecla -no-wrap''`. Sobre el valor por defecto únicamente se ha añadido la opción `-no-wrap` al final.
    - No es necesario modificar nada mas. Pulsar *Apply* y *Ok*.

Ahora ya tenemos el espacio de trabajo preparado para escribir especificaciones y ejecutarlas.

# 4. Como definir una especificación algebraica y ejecutarla

En este apartado vamos a aprender a definir una especificación y ejecutarla. Para ello utilizaremos como ejemplo una especificación de los números complejos con las operaciones de suma y resta.

Para poder definir una especificación:

- Ejecutar Eclipse. Seleccionar un espacio de trabajo (directorio en el que se guardan los proyectos que se realicen).
- Primero hay que crear un proyecto. En la opción *file* seleccionar *new* y a continuación *other*. Desplegar *General* y elegir *Project*. Elegir un nombre para el proyecto, si se va a utilizar el *testing* el proyecto se debe denominar `Testing` (Atencion a las mayúsculas), si no se va a utilizar el *testing* el nombre del proyecto es indiferente, p.e. `COMPLEJOS`. Pulsar *Finish*. Observad que aparece el nombre del proyecto en la ventana *Package*.
- En este momento, si tenemos el fichero en formato texto ya creado, podemos seleccionar en el menú *File*, la opción de *import*. En el desplegable seleccionar *Archive File* y pulsar *Next*. Seleccionar el archivo a importar y el proyecto en el que se debe importar. Un inconveniente de esta forma de añadir archivos a proyectos es que solo se admiten ficheros comprimidos. Otra forma de añadir un fichero ya creado es crear un nuevo fichero en el proyecto (como se explica a continuación) y copiar el contenido del que ya existe.
- Si no tenemos el archivo ya creado, creamos un fichero Maude en el proyecto. Para ello seleccionar en el menú *File* la opción *Other* y aquí en la opción *Maude* elegir *Maude file*. En el apartado *Container* debe seleccionarse el proyecto al que queremos que pertenezca este archivo. En este caso,

seleccionad el proyecto `Testing` o `COMPLEJOS`. Escribid un nombre para el archivo, por ejemplo `Complejos`. La extensión del archivo debe ser `.maude`. Si el nombre del archivo no lleva la extensión no se interpretará como un archivo de Maude. Pulsar *Finish*.

- En el fichero `Complejos` escribid la especificación, utilizando el editor situado en la ventana central:

```
(fmod COMPLEJOS is
  including INT .
  sort Complejo .
  op C : Int Int -> Complejo [ctor] .
  op suma : Complejo Complejo -> Complejo .
  op resta : Complejo Complejo -> Complejo .
  vars A1 A2 A3 A4 : Int .
  eq [suma] : suma(C(A1,A2),C(A3,A4)) = C(A1 + A3, A2 + A4) .
  eq [resta] : resta(C(A1,A2),C(A3,A4)) = C(A1 - A3, A2 - A4) .
endfm)
```

  Al escribir la especificación hay que tener cuidado con los espacios en blanco, los puntos y las mayúsculas y minúsculas ya que en Maude tienen significado.

  Con esta especificación los números complejos se representan con la constructora `C`. Las operaciones de suma y resta se denominan `suma` y `resta` respectivamente. Este curso no se utilizarán operaciones con notación infija debido a las restricciones de la herramienta de *testing*.

  Salvad la especificación al terminar de escribirla, utilizando el menú *File* de Eclipse o el icono de la barra de herramientas.

- Ejecutad Maude pulsando en el botón de la consola. Vereis que os sale un mensaje *Welcome to Maude* en la consola y que aparece una línea de comandos.

- Ahora hay que cargar el módulo que hemos escrito en el sistema. Para ello se puede utilizar la opción *Maude* del menú con *Send to Maude* o bien el botón de la la barra de herramientas. Lo que se envía al sistema Maude con esta opción es lo que se encuentra en la ventana activa, por ello hay que asegurarse de que la ventana activa es la del editor en la que se encuentra la especificación que queremos ejecutar.

- Si hemos cometido algún error al escribir el módulo, y este no se ajusta a la sintaxis de Maude, aparecerá un mensaje en la consola, donde se nos dice (mas o menos) en que consiste el error. Arregladlo y volved a introducir el módulo.

- Cuando el módulo se carga en el sistema sin problemas, el sistema lo indica. Cuando se introduce un módulo que ya estaba en el sistema, por haberse introducido anteriormente se produce un mensaje indicando que el módulo ha sido redefinido.

- Ahora podemos reducir términos en esta especificación. Escribid, por ejemplo, el término `suma(C(3,4),C(7,2))` en el editor precedido por el comando `red` y acabado en `.`, todo ello entre paréntesis (En *Full Maude* los comandos deben escribirse entre paréntesis) (`red suma(C(3,4),C(7,2))` .). A continuación seleccionar la línea y darle al botón *Send selection to Maude* del menú *Maude*. En la consola aparece la respuesta del sistema. Si el término es correcto el sistema contestará *result Complejo:* `C(10,6)`.

  Cada vez que se cargue el módulo en el sistema se reducirán todos los términos que aparezcan en el mismo precedidos de la palabra reservada `red`. Para evitarlo se pueden comentar estas líneas. Los comentarios están precedidos por tres guiones (`---`). Los guiones no deben seleccionarse al reducir los términos.

- Otra forma de reducir términos es introducir el comando en la caja existente en la parte inferior de la consola.

- Los términos se reducen en el último módulo que se cargó en el sistema (módulo activo). Para saber que módulo se encuentra activo cuando queremos reducir un término se puede introducir el comando `show module .` que nos mostrará el módulo activo.
- Si al cargar un módulo o reducir un término aparece la palabra `executing...` en la parte superior de la consola es que el sistema está esperando algún símbolo que no se ha introducido, o bien que se está realizando una reducción infinita. Para finalizar el programa pulsar el botón de abandonar Maude (Quit).

# 5. Conceptos básicos de Maude

En este apartado solo se introducen los conceptos y la sintaxis de Maude necesarios para poder definir especificaciones de tipos de datos sencillos. El lenguaje proporciona muchas mas facilidades de las que aquí se van a introducir. Los estudiantes interesados en escribir especificaciones mas complejas o en utilizar una sintaxis mas completa pueden consultar el manual del lenguaje.

Para comentar una línea de un fichero Maude se escriben tres guiones al iniciar la línea.

## 5.1. Módulos funcionales no parametrizados

Las especificaciones algebraicas se escriben en Maude en *módulos funcionales*. Un ejemplo de módulo funcional es el que se ha escrito en el apartado anterior para los números complejos.

- Los módulos funcionales empiezan con la palabra reservada `fmod`, el nombre del módulo y la palabra reservada `is`.
- A continuación aparecen otros módulos que se necesitan en la definición de esta especificación. El nombre del módulo que se importa va precedido de la palabra `including`. Existen otras formas de importar módulos para las que se utilizan las palabras reservadas `protecting` y `extending`. La diferencia entre ellas está fuera del ámbito del curso. Los valores booleanos están incluidos por defecto y no es necesario importarlos explícitamente.
- A continuación, precedido de la palabra `sort` aparece el nombre del tipo de datos que se va a definir seguido de un blanco y un punto. En caso de definirse varios tipos de datos se darían de uno en uno precedidos cada uno por la palabra `sort`.
- Cuando existe una relación de subtipo (todos los valores de un tipo pertenecen al otro tipo) entre dos tipos se indica con la palabra reservada `subsort`. Por ejemplo `subsort Nat <Int .`
- La definición de cada operación va precedida de la palabra `op`. A continuación aparece el nombre de la operación en forma prefija o infija, dos puntos entre blancos y los tipos de los argumentos. En el ejemplo, el tipo `Int` que aparece en el módulo de los números complejos esta definido en el módulo `INT` que hemos importado (ver los módulos predefinidos en la Sec. 5.4). Después de los argumentos se escribe una flecha (→) consistente en un guión y un signo mayor y el tipo del resultado de la operación. La declaración acaba en un blanco seguido de un punto. Cada operación se definirá en una línea y siguiendo esta sintaxis. Las constructoras llevan el identificador `[ctor]` antes del punto final.

  Las operaciones se pueden escribir en forma prefija o infija. Para escribir una operación en forma prefija escribimos unicamente su nombre. Por ejemplo `op sumar : Int Int → Complejo`. Los términos que utilizan esta operación la escriben como `sumar(A,B)`. Para escribir una operación en forma infija, se escribe el símbolo subrayado en el lugar donde deben aparecer los argumentos. Por ejemplo `op _+_ : Int Int → Complejo`. Los términos que utilizan esta operación la escriben como `A + B`. Las variables se separan del símbolo de operación con un blanco. Como se indicó anteriormente este curso solo se utilizarán operaciones en notación prefija.

- Si en la definición de las ecuaciones hacen falta variables, estas aparecen precedidas de la palabra `var` (o `vars` si se definen varias variables del mismo tipo). Si se declaran varias variables estas van separadas por blancos. A continuación se escriben dos puntos, el tipo de las variables y un punto. Todo ello separado por uno o mas blancos.
- Por último se escriben las ecuaciones. Van precedidas de la palabra `eq` seguida de un identificador entre corchetes y dos puntos. A continuación los dos términos separados por el signo `=` y finalizan con un blanco y un punto.

  Las ecuaciones condicionales van precedidas de la palabra `ceq`, el identificador y los dos puntos, a continuación la ecuación y la palabra `if` seguida de la condición. Las condiciones se escriben comprobando si dos términos son iguales (`==`) diferentes (`=/=`) y uniendo términos booleanos con las conectivas `and`, `or` y `not` .

  El identificador de las ecuaciones debe ser único en el módulo.
- El módulo termina con la palabra `fmod`.
- Los módulos van entre paréntesis para indicar al sistema que están escritos en *Full Maude*.

## 5.2. Parametrización

La especificación de un módulo parametrizado es muy similar a la de los módulos sin parámetros.
La siguiente especificación define el TAD de las pilas.

```
(fmod PILA{X :: TRIV} is
  sort Pila{X} .
  op errorPila : -> [Pila{X}] .
  op errorElemPila : -> [X$Elt] .
  op pila-vacia : -> Pila{X} [ctor] .
  op apilar : X$Elt Pila{X} -> Pila{X} [ctor] .
  op desapilar : Pila{X} -> Pila{X} .
  op cima : Pila{X} -> X$Elt .
  op es-pila-vacia : Pila{X} -> Bool .
  var P : Pila{X} .     var E : X$Elt .
  eq [desapilar1] : desapilar(pila-vacia) = errorPila .
  eq [desapilar2] : desapilar(apilar(E,P)) = P .
  eq [cima1] : cima(pila-vacia) = errorElemPila .
  eq [cima2] : cima(apilar(E,P)) = E .
  eq [pilaVacia1] : es-pila-vacia(pila-vacia) = true .
  eq [pilaVacia2] : es-pila-vacia(apilar(E,P)) = false .
endfm)
```

```
(view vInt from TRIV to INT is
  sort Elt to Int .
endv)
```

```
(fmod PILA-ENTEROS is
    including PILA{vInt} .
endfm)
```

- El parámetro se define a continuación del nombre del módulo entre llaves. Aparece en primer lugar el nombre dado al parámetro en el módulo. A continuación entre blancos `::` y por último el nombre de la teoría que define el parámetro. La teoría `TRIV` la proporciona el sistema Maude y consiste en la definición de un tipo denominado `Elt`. Si se quiere definir más de un parámetro, estos se separan con comas, tanto en la cabecera como cada vez que nos referimos al tipo de datos.
- Cada vez que nos referimos en la especificación al tipo `Pila` debemos anotar a continuación entre llaves el nombre del parámetro.
- Cuando nos referimos en la especificación a los datos del parámetro escribiremos el nombre del parámetro, seguido del signo $ y del nombre del tipo de datos del parámetro. Por ejemplo `X$Elt`.

- Una vez definido el módulo paramétrico debe definirse una vista que relacione el parámetro con una especificación.
    - La definición de una vista comienza con la palabra reservada `view`, a continuación se escribe el nombre de la vista, la palabra reservada `from`, el nombre de la teoría dada como parámetro, la palabra `to`, el nombre de la especificación que queremos asociar al parámetro y la palabra `is`
    - La línea siguiente relaciona el tipo del parámetro, en nuestro caso `Elt` con el tipo de la especificación, en nuestro caso `Int`.
    - Si la teoría que estamos utilizando define operaciones, es necesario relacionar en la vista las operaciones del parámetro con las operaciones de la teoría. Esto se hace de forma similar a la relación entre los tipos: `op suma to _+_`.
    - La vista acaba con la palabra `endv`.
- Por último se define el módulo instanciado, que es el que utilizaremos para reducir términos. Se declara un módulo funcional, y se incluye la especificación de las pilas instanciada con el nombre de la vista.

## 5.3. Módulos predefinidos

Los módulos predefinidos se encuentran en el fichero `prelude.maude` que se proporciona con el sistema.

Para poder utilizar los tipos definidos en estos módulos, excepto BOOL, debe importarse el módulo correspondiente.

Tenemos entre otros:

- `BOOL` define los valores `true` y `false` y las operaciones `_and_`, `_or_`, `_xor_`, `not_`, `_implies_`.
- `NAT` define los números naturales. Estos números pueden utilizarse en notación decimal. Algunas de las operaciones que proporciona son: `_+_`, `sd : Nat Nat → Nat` para la diferencia, y `_*_`.
- `INT` define los números enteros. Estos números pueden utilizarse en notación decimal. Algunas de las operaciones que proporciona son: `-_`, `_+_`, `_-_`, `_*_`, `_quo_` calcula el cociente de la división entera, `_rem_` calcula el resto de la división entera.
- `STRING`. En este módulo se definen los tipos `String` y `Char`. Tanto las cadenas de caracteres como los caracteres se escriben entre comillas dobles, p.e. "a" o "Hola". Algunas de las operaciones que proporciona este módulo son: `_+_` para la concatenación, y `lenght` para calcular la longitud de una cadena.

El resto de los módulo predefinidos y las operaciones que no se han nombrado aquí pueden consultarse en el manual.

## 5.4. Teorías predefinidas

Existen varias teorías definidas por defecto en el fichero `prelude.maude`. Estas teorías pueden utilizarse sin necesidad de hacer ninguna declaración.

1. `TRIV` declara un tipo `Elt` .
2. `STRICT-WEAK-ORDER` declara el tipo `Elt` y una operación `_<_`.
3. `TOTAL-PREORDER` declara el tipo `Elt` y una operación `_<=_`.
4. `DEFAULT` declara un tipo `Elt` y una constante `0`.

Aunque existen muchas vistas predefinidas en el archivo `prelude.maude` es preferible durante este curso que cada alumno defina las vistas que necesite.